

# Formal Verification of Data-Obliviousness in Hardware

**MBMV'23 – March 24, 2023**

**Technische Fakultät, Universität Freiburg**

Lucas Deutschmann, Johannes Müller, Mo Fadiheh, Dominik Stoffel, Wolfgang Kunz  
Dept. of Electrical & Computer Engineering, RPTU Kaiserslautern-Landau



- “Cambrian explosion” of confidential computing technologies
  - Cryptographic methods, fully homomorphic encryption
  - Trusted execution environments, secure enclaves
  - Microarchitectural security defenses
  - ...
- All of these target **confidentiality**, for different threat models



They all **fail** unless **data-oblivious computing** is supported throughout all levels of the system stack

## *Data-Oblivious Computing:*

- Runtime,
- resource usage and
- memory access patterns

of the program are independent of confidential data.

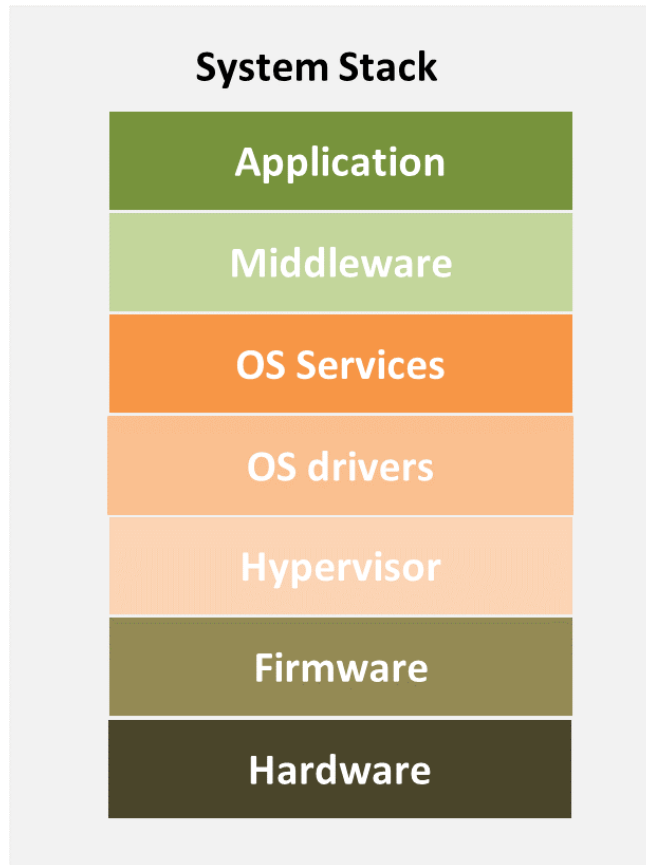
```
def check_key(user_input):  
    if user_input != SECRET_KEY:  
        raise Exception("Wrong key provided!")
```

## Problem:

String comparison operator compares one byte at a time, stops as soon as a mismatch is found

- Not data-oblivious!
- Attacker can deduce the secret byte by byte, by measuring the runtime

# A challenge across the system stack



## ➤ “Constant-Time” Programming

➤ *Writing programs such that their runtime and resource usage is independent of confidential information*

## ➤ Contributions from the SW community

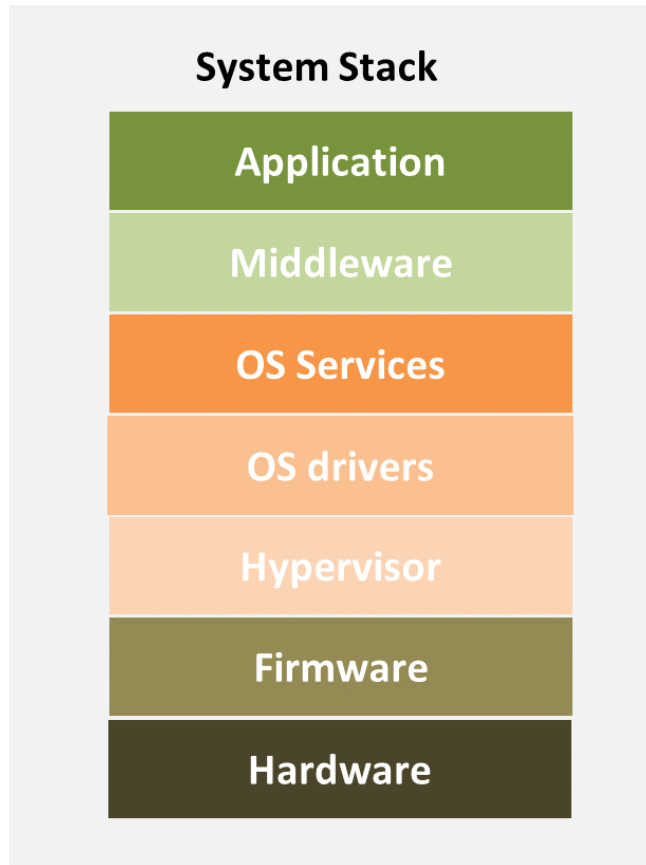
➤ Open-source libraries [BearSSL, <https://bearssl.org/>]

➤ DSLs [Cauligi'17]

➤ Verification Tools [Almeida'18]

➔ Can SW (alone) fix the problem?

# SW fixes are insufficient!



## ➤ “Opening Pandora’s Box” [Vicarte’21]

➤ 7 microarchitectural optimizations that undermine the constant-time paradigm

## ➤ Threat is real!

➤ Data memory-dependent prefetchers recently found in Apple A14, M1 and M1 Max devices

➤ Security breach found [<https://prefetchers.info/>]

➔ How can we restore the trust in HW?

- Verification of “constant-time” SW [e.g., Cauligi’20]
- Guarantees regarding HW/SW relationship
  - Data-Oblivious ISA Extension [Yu’20]
  - HW/SW Contracts [Guarneri’20]
  - RISC-V Cryptography Extension [<https://github.com/riscv/riscv-crypto>]
- Almost no formal verification for data-oblivious HW in RTL
  - Clepsydra [Ardeshiricham’17]
  - XENON [Gleissenthall’21]

# HW Root-of-Trust: Requirements


- Data-obliviousness of transient instruction execution (resilience against Spectre, Meltdown, ...)
  - **Our previous work:** *Unique Program Execution Checking (UPEC)* [DATE'19, DAC'20, DAC'21, TC'22]
  - Exhaustive and scalable approach ✓
- **HW primitives for data-oblivious operations:**  
Operation execution is independent of timing and resource usage
  - Open problem !
  - **This work:** *UPEC for data-independent timing (UPEC-DIT)* [DAC'22]



# UPEC for Data-Independent Timing

## UPEC-DIT:

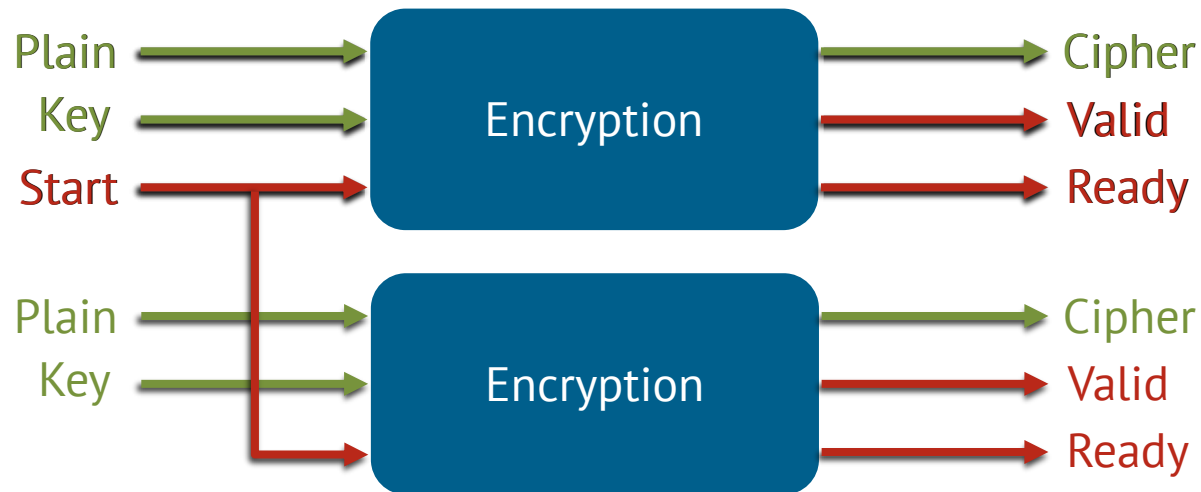
Formal approach for detecting data-dependent timing in RTL designs

- Qualifies a microarchitectural ISA implementation
  - Determines the **data-oblivious ISA subset** of a given CPU
  - Provides **guarantees** for “constant-time” programming
- Leverages state-of-the-art commercial property checking
- Scalable to realistic designs (~700k state bits)
-  Found **violations of data-obliviousness** in security-conscious designs

- Interval Property Checking (IPC) on 2-safety computational model
- Major extension of UPEC
  - Previously: Verifies confidentiality of data-at-rest
  - **UPEC-DIT**: Verifies confidentiality of data-in-transit
    - Tolerates legal propagations of secret data
    - Detects data-dependent variations of the control behavior
- ➔ Create an implicit representation of the HW's control behavior
  - In terms of semi-automatically determined set of control signals

# Easy case: UPEC-DIT for Functional Units

FUs can be treated as **black boxes**: control signals are given by I/O spec



- Separate **control** and **data**
- Generate 2-safety model
- Formulate property

```
assume:  
  at t: State_Equivalence();  
prove:  
  during t..t+k: Control_Output_Eq();
```

# UPEC-DIT for Functional Units

Design	Data-Ind. Timing	#States	Proof Time (s)	Max. Mem (MB)
BasicRSA-T100	X	532	< 1	589
SHA1	✓	911	< 1	306
SHA256	✓	1103	< 1	296
SHA512	✓	2162	< 1	329
AES1	✓	2472	4	994
AES2	✓	554	< 1	819
FWRISCV MDS-Unit	(!)	331	< 1	596
ZipCPU Div-Unit	X	142	11	1347
CVA6 Div-Unit	X	209	< 1	580

# Hard Case: UPEC-DIT for Processors

- Distinguish between legal and illegal timing variabilities w.r.t. “constant-time” programming
  - **ISA-visible** timing variations are **legal** (e.g., stalls due to dependencies between instructions)
  - **ISA-invisible**, operand value-dependent timing variations are **illegal**
- Global analysis of I/O behavior of HW is not tractable
  - White-box approach necessary:
  - Control flow must be represented in terms of **internal** control signals
- Must consider instructions under **any** pipeline context

# Hard Case: UPEC-DIT for Processors

## Solution:

- Security-critical timing behavior is determined by a small set of control signals
- Which control signals must be considered?
  - Iterative procedure
  - Starts with *all* state-holding signals
  - Refines property by analyzing propagation alerts
- 2-safety computational model with symbolic starting state, by construction, excludes legal timing variations, e.g., RAW hazards

- *State\_Equivalence()*:
  - Constrains state bits to be equal in the two instances
  - **Except** for the **operand sources**
    - usually, the register file and forwarded operands
- $t_{wb}$  denotes time point when instruction under verification (IUV) finishes execution

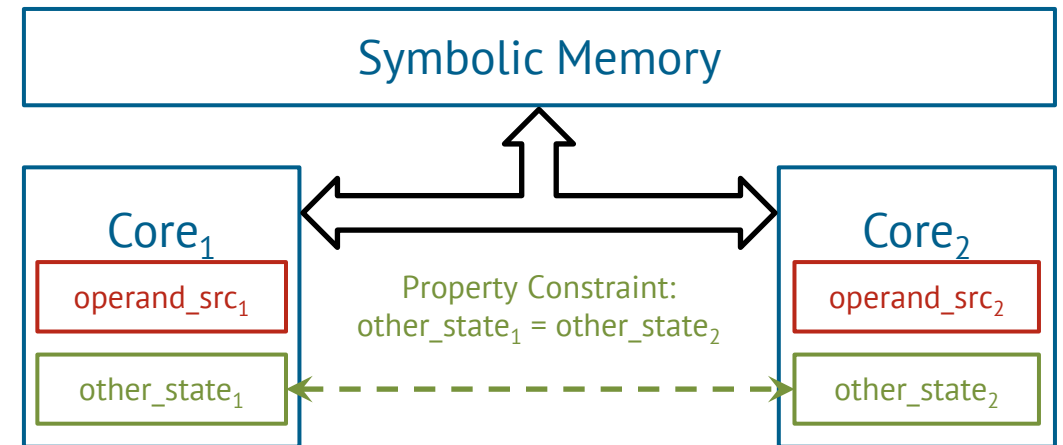
assume:

at t: *State\_Equivalence()*;

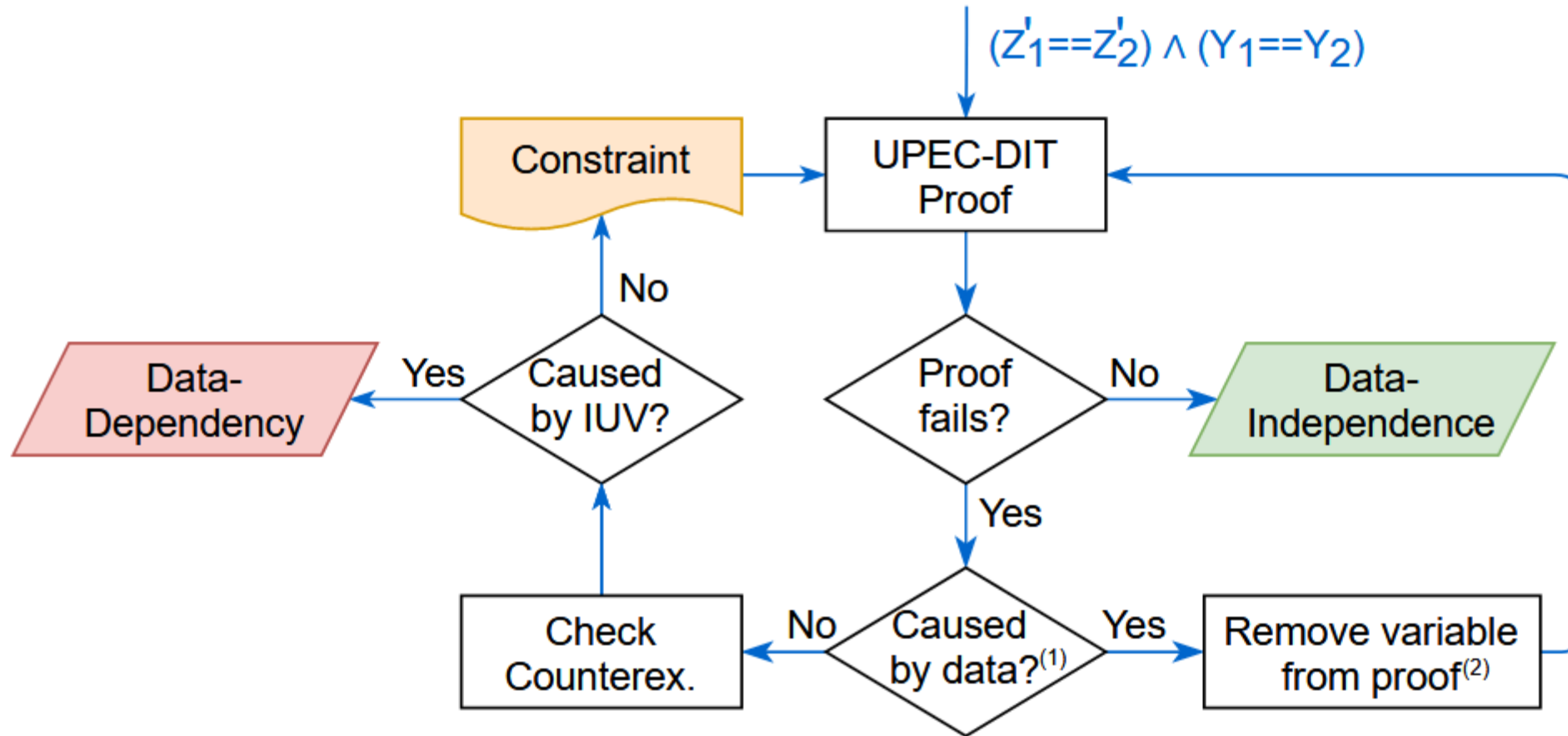
at t: *IUV\_in\_ID\_Stage(type)*;

prove:

during  $t..t_{wb}$ : *Control\_Equality()*;



# UPEC-DIT for Processors



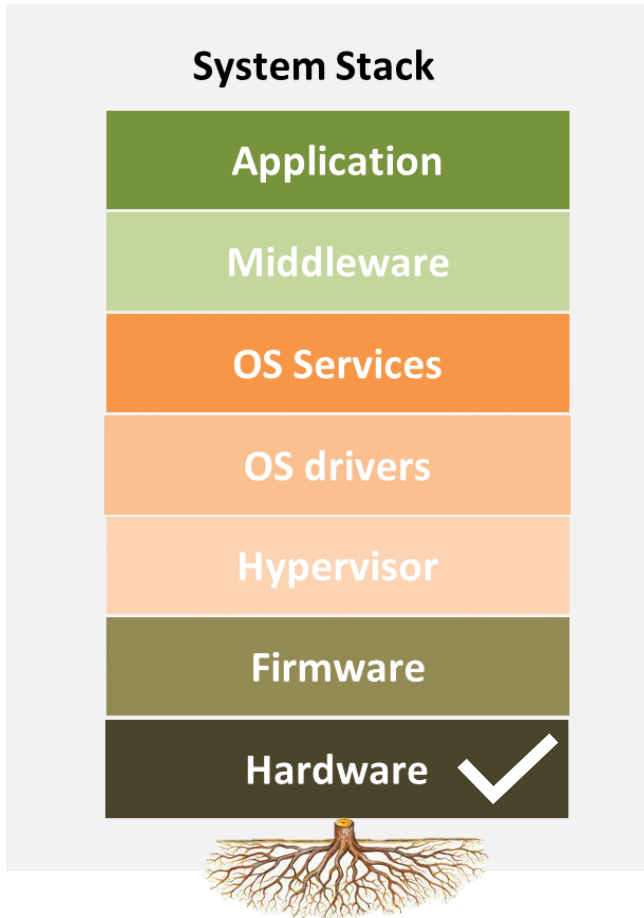


# UPEC-DIT for Processors

	FWRISCV	IBEX	IBEX (+DIT)	SCARV	CVA6
I-Type Arith.	✓	✓	✓	✓	✓
R-Type Arith.	✓ ✗	✓ ✓	✓ ✓	✓ ✓	✓ ✓
Multiplication	✓ ✓	✓ ✗	✓ ✓	✓ ✓	✓ ✓
Division	✓ ✓	✓ ✗	✓ ✓	✓ ✓	✗ ✗
Load	(!)	(!)	(!)	✗	✗
Store	(!) ✓	(!) ✓	(!) ✓	✗ ✗	✗ ✓
Jump	(!)	✓	✓	(!)	✗
Branch	✗ ✗	✗ ✗	✓ ✓	✗ ✗	✗ ✗
#State Bits	3086	1019	1021	2334	682849
Average Time	3s	2min	4min	3min	1h 36min
Worst-Case Time	4s	5min	7min	8min	3h 7min
Max. Mem (GB)	1.7	4.5	4.3	2.1	11.9

# Berkeley Out-of-Order Machine (BOOM)

- Superscalar RISC-V processor with FP support, a deep 10-stage pipeline and out-of-order execution
- Results:
  - Proved data-obliviousness for I-type arithmetic, R-type arithmetic, multiplication (and FP arithmetic)
  - Data-dependent timing in Int. Division, FP Division and FP Sqrt
  - Properties take up to 20 hours
- **Current work:** Develop an inductive property for better scalability



- **UPEC-DIT** detected several **unknown violations** of data-obliviousness
- **Scalable** and largely **automated** for RTL designs
  - Instruction-level granularity: Well-defined interface with **security guarantees** for the entire system stack
  - Current work: Extend to and inductive property to ensure scalability even for complex systems
- Closes important gap in making HW a root-of-trust for entire system stack

# Thank you for your attention!

**Many thanks to many collaborators!**

Jörg Bormann, Anna Lena Duque-Antón, Wolfgang Ecker, Mohammad Rahmani Fadiheh, Jason Fung, Tobias Jauch, Wolfgang Kunz, Johannes Müller, Dino Mehmedagić, Subhasish Mitra, Sayak Ray, Philipp Schmitz, Stian Gerlach Sørensen, Dominik Stoffel, Alex Wezel

The reported research was partly supported by BMBF ZuSE (*Scale4Edge*), 16ME0122K-16ME0124, by DFG SPP *Nano Security*, KU 1051/11-1, and by Intel (*Scalable Assurance*).

- Mohammad R. Fadiheh, Dominik Stoffel, Clark Barrett, Subhasish Mitra, Wolfgang Kunz: “Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking”, *Design Automation and Test in Europe (DATE)*, 2019.
- M. R. Fadiheh, J. Müller, R. Brinkmann, S. Mitra, D. Stoffel, and W. Kunz: “A formal approach for detecting vulnerabilities to transient execution attacks in out-of-order processors,” *57th IEEE/ACM Design Automation Conf. (DAC’20)*, 2020.
- J. Müller, Mo Fadiheh, A. Duque Anton, T. Eisenbarth, D. Stoffel and W. Kunz: “A Formal Approach to Confidentiality Verification in SoCs at the Register Transfer Level“, *58th IEEE/ACM Design Automation Conf. (DAC’21)*, Dec. 2021.
- L. Deutschmann, J. Müller, Mo Fadiheh, D. Stoffel and W. Kunz: “Towards a Formally Verified Hardware Root-of-Trust for Data-Oblivious Computing “, *59th IEEE/ACM Design Automation Conf. (DAC’22)*, July 2022.
- Mohammad R. Fadiheh, Alex Wezel, Johannes Müller, Jörg Bormann, Sayak Ray, Jason M. Fung, Subhasish Mitra, Dominik Stoffel, Wolfgang Kunz: “An Exhaustive Approach to Detecting Transient Execution Side Channels in RTL Designs of Processors”, *IEEE Transactions on Computers*, 2022 (in preview).

Github: <https://github.com/mofadiheh/upec-boom-verification-suite>